BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

4

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-208982
(43)Date of publication of application : 26.07.2002

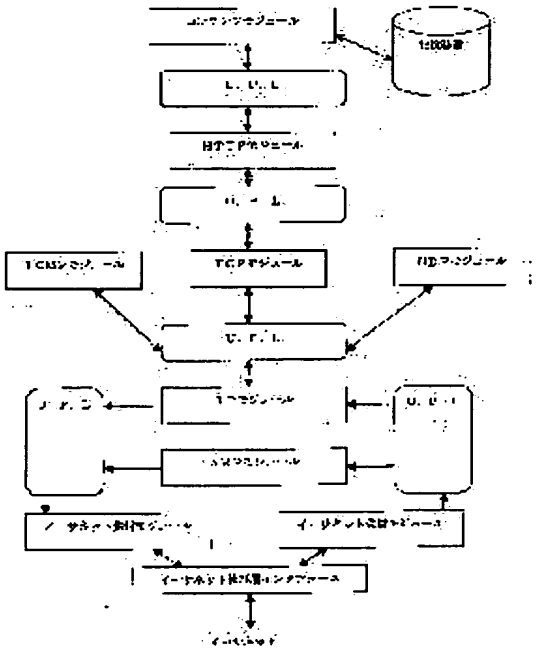
(51)Int.Cl. H04L 29/06
H04L 12/56

(21)Application number : 2000-404731
(22)Date of filing : 28.12.2000

(71)Applicant : E TREES JAPAN:KK
(72)Inventor : FUNADA SATOSHI

(54) APPARATUS OF PROCESSING COMMUNICATION PROTOCOL AT HIGH SPEED BY REPLACING SOFTWARE WITH HARDWARE

(57)Abstract:
PROBLEM TO BE SOLVED: To obtain an information communication processor in internet environment in which high speed processing is attained by replacing communication processing software with hardware.
SOLUTION: A communication protocol is processed at high speed by replacing communication processing software with hardware through (1) a communication processor provided/not provided with a header and (2) a UPL(Universal Protocol Line).



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（Ａ）**　(11)特許出願公開番号
特開2002－208982
（P2002－208982A）
(43)公開日　平成14年７月26日(2002.7.26)

(51)Int.Cl.⁷		識別記号	FI		テーマコード（参考）
Ｈ０４Ｌ	29/06		Ｈ０４Ｌ	12/56	３００Ａ　５Ｋ０３０
	12/56	３００		13/00	３０５Ａ　５Ｋ０３４

審査請求　未請求　請求項の数８　書面　（全 12 頁）

(21)出願番号　特願2000－404731(P2000－404731)

(22)出願日　平成12年12月28日(2000.12.28)

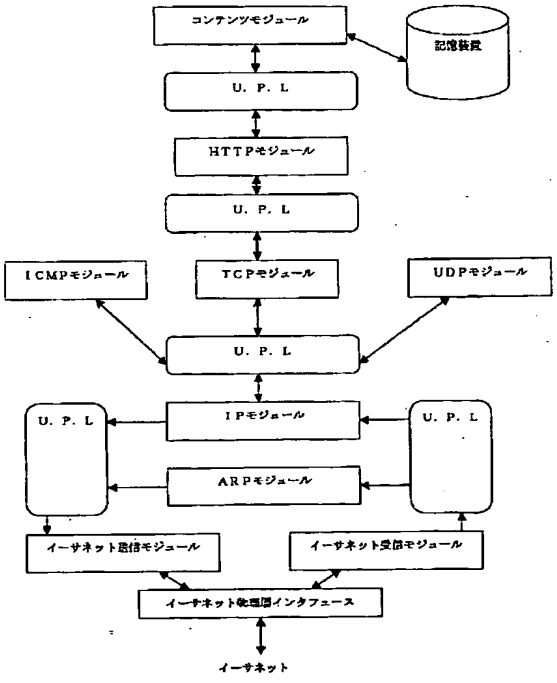
(71)出願人　501053026
株式会社イーツリーズ・ジャパン
東京都渋谷区広尾１丁目11番４号　共立ビル501

(72)発明者　船田　悟史
東京都　目黒区　南三丁目９番17号　ストークハイツＹＡＭＡＧＡＴＡ101

Ｆターム(参考)　5K030 GA01 HA08 JA05 LE06
5K034 AA01 BB06 FF03 HH01 HH02
MM25

(54)【発明の名称】　ソフトウェアをハードウェアに置き換えることによって通信プロトコルを高速処理する装置

(57)【要約】
【課題】　インターネット環境における情報通信処理装置において、通信処理をソフトウェアからハードウェアに置き換えることによって高速処理を可能とする。
【解決手段】　ヘッダの完備・不完備化による通信処理装置（１）とＵＰＬ（Ｕｎｉｖｅｒｓａｌ　Ｐｒｏｔｏｃｏｌ　Ｌｉｎｅ）（２）によって、通信処理ソフトウェアをハードウェアに置き換え、通信プロトコルの高速処理を行う。



1

【特許請求の範囲】

【請求項１】　インターネット環境における情報通信処理装置において、これまでの一貫したＣＰＵ処理に変わる、ヘッダの完備化・不完備化による通信処理装置（１）とＵＰＬ（Ｕｎｉｖｅｒｓａｌ　Ｐｒｏｔｏｃｏｌ　Ｌｉｎｅ）装置（２）によって、通信情報処理装置の通信処理ソフトウェアをハードウェアに置き換え、通信プロトコルの高速処理を行う。この結果、ワイヤースピードで通信処理をすることを可能とする。

【請求項２】　請求項１に記載したヘッダの完備化・不完備化による通信処理装置とは、外部入力からの通信プロトコル情報を共通化・抽象化（３）することによって処理を迅速にする方法である。内部処理において情報流通に必要なものだけを故意的に共通化・抽象化し、不完備化することにより（４）、プロトコル処理を高速化することが可能となる通信処理装置。

【請求項３】　請求項２に記載した不完備化の具体例として、ＵＮＩＸ（登録商標）、Ｌｉｎｕｘ、Ｗｉｎｄｏｗｓ（登録商標）等のインターネットにおける情報交換手段として広く用いられているＩＰプロトコル・ＴＣＰプロトコル・ＨＴＴＰプロトコルによる処理内容を次に示す。現在通信における説明として広く用いられているＯＳＩ７階層モデル（５）において、通信プロトコルは第１階層から第７階層へとレイヤーに分割されて処理が行われている。このレイヤーの中では、外部入力からのヘッダ情報を共通化・抽象化する処理が行われる。外部においてはヘッダ情報としては不完備になるものの、内部レイヤーにおいて共通化・抽象化したヘッダ情報を生成する装置。

【請求項４】　ＵＰＬ装置（６）とは、関数呼び出しによって各レイヤー間においてヘッダ・データ情報のやり取りと処理がおこなわれているソフトウェアを構成する複数のモジュール間において前記ヘッダの不完備化による通信処理装置によって外部情報を共通化・抽象化した情報を一つの構成要素として、モジュール間でやり取りするための装置。

【請求項５】　本発明は、前記ＵＰＬ装置を用いて、各レイヤー（物理層・データリンク層・ネットワーク層・トランスポート層・セッション層・プレゼンテーション層・アプリケーション層）７層をハードウェアで入出力処理を行う装置。

【請求項６】　出力に関して、前記ＵＰＬ装置とヘッダの完備化による情報処理装置を用いて、ハードウェア上で不完備ヘッダ情報を完備ヘッダ情報に戻し（７）、必要に応じて情報テーブル（８）を参照しながら、必要な情報をヘッダと一緒に要求された装置に迅速に送ることができる装置。

【請求項７】　入力部（９）と出力部（１０）によって、インターネットにおける通信プロトコル処理とデータ処理をワイヤースピードで処理することができることを可能とした通信処理装置。

2

【請求項８】　請求項７における装置に記憶装置（１１）、ユーザー入出力装置を対応させて、サーバ・ルータ・クライアント装置等に応用することができる。具体例として、サーバにおいては、Ｗｅｂサーバをはじめとしてメールサーバ、ＦＴＰサーバ、ＤＮＳサーバ、ニュースサーバ等の装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、メモリ・入力／出力装置・中央処理ユニット間の情報・他の信号の相互接続・転送を統一アクセス制御理論によりＷｅｂサーバ・メールサーバ・ＦＴＰサーバ・ＤＮＳサーバを実装レベルで実現する方式および方法に関する。

【０００２】

【従来の技術分野】従来、インターネットにおけるＷｅｂサーバを構築する場合にＣＰＵを一つまたは数個使い一貫処理をさせている現状である。

【０００３】

【発明が解決しようとする課題】従来の処理をＣＰＵに依存する方法を用いた場合には、集中的にサーバに対してアクセスが世界中のユーザーからかかると、処理しきれずにサーバが急停止しサービスを利用できなく、結果としてインターネットインフラに対しての信用の低下を招きうるものであった。また、これからのＡＤＳＬ・ＳＤＳＬ及び光ファイバー等の拡充にともない、乗数倍的にサーバに対して負荷がかかることが容易に予想される。

【０００４】従って、本発明の目的は、乗数倍的にサーバに対して負荷がかかっても、情報を処理しきれなく、結果として、処理を停止してしまうことを回避し、ギガビットレベルで瞬時に処理し、インフラとしてのインターネットに対して信頼をもたせることである。このことにより、これからの情報化社会に対して社会貢献的役割を担える。

【０００５】

【課題を解決するための手段】このような目的を達成するために、請求項１の発明は、ヘッダの不完備化による情報処理装置とＵ．Ｐ．Ｌ（ＵｎｉｖｅｒｓａｌＰｒｏｔｏｃｏｌ　Ｌｉｎｅ）理論による解決方法を必要とする。

【０００６】請求項１に記載のヘッダの不完備化による情報処理装置とは、外部入力からの情報を共通化・抽象化することによって処理を迅速にする方法である。この点、もともとのデータ量と比較すると情報量は減らされているが、内部処理において情報流通に必要なものだけを故意的に共通化・抽象化し、不完備化することにより、プロトコル処理を高速化することが可能となる。この方法を使って、情報処理装置におけるプロトコル処理を円滑に処理することができる。

【０００７】Ｕ．Ｐ．Ｌ（Ｕｎｉｖｅｒｓａｌ　Ｐｒｏｔｏｃｏｌ　Ｌｉｎｅ）理論とは、関数呼び出しによって情報のやり取りと処理がおこなわれているソフトウェアを構成する単一または複数のモジュール間において外部からの既存の情報と内部において前記、ヘッダの不完備化による情報処理装置によって外部情報を共通化・抽象化した情報を一つの構成要素として、モジュール間をやり取りするための方法である。

【０００８】この二つの理論を用いて基板上においてギガビット単位の情報処理を可能とする。

【発明の実施の形態】以下、図面を参照して本発明の実施形態を詳細に説明する。

【０００９】今日開発されているソフトウェアの規模は非常に大きなものであり、一人のプログラマーが単独で開発を完了することは非常に困難である。そのため、一つのソフトウェアの開発に複数人のプログラマーが互いに協力しあい、開発が進められている。

【００１０】複数のプログラマーが一つのソフトウェアを開発する場合、有効な開発手法がいくつかしられている。もっとも一般的な方法として、巨大なソフトウェアを部分に分割し、その各部分をそれぞれプログラマーに割り当て、独立して開発を行う。最終的に、部分を一つに結合して、ソフトウェアを完成させるという方法である。

【００１１】一般にソフトウェアは、複数の機能の組み合わせによって実現される。図７のように、ワープロソフトにおいては、ユーザーが入力した文字を漢字に変換する機能、入力された文字を文面に構成する文章編集機能、文章をファイルに保存したり、ファイルから文章を読み出す機能、文章に絵を書き入れる機能等の機能がある。ワープロソフトの機能がますます高度化するにともない、その実現が非常に困難になる。

【００１２】ところが、機能をそれぞれ独立に開発し、独立に開発されたソフトウェアを結合して、ワープロソフトという形にすることで、プログラマーは個別機能の開発に力を集中することができ、最終的な製品の完成度を高めることも貢献できる。

【００１３】しかし、ソフトウェアを部分に分解し、独立して開発するにあたって、問題もいくつか提起されている。ソフトウェアをどのように部分に分解するか、部分どうしをどのように結合するかという問題である。

【００１４】これらの問題に対して、プログラミング言語に応じてさまざまな解決方法が提案されている。例にとると、アセンブリ言語、ＢＡＳＩＣ言語、サブルーチン、構造化プログラミング、Ｃ言語、関数呼び出し、分割コンパイル、リンク機能、Ｃ＋＋言語、Ｊａｖａ言語、オブジェクト指向プログラミングなどが挙げられる。これらの手法を用いて、プログラマーは自分の担当するソフトウェアの一部分を開発することが行われる。

【００１５】本発明においては、言語等の関数呼び出しにより部分開発されるソフトウェアの一部もしくは全体をハードウェアによって実現するものである。

【００１６】次に、本発明によりハードウェア化できるソフトウェアかできるソフトウェアの構成法を説明する。ソフトウェアの一つの機能を実現する部分をモジュールと呼ぶことにする。ソフトウェアは一つ以上のモジュールから構成され、モジュールの間は関数呼び出しによって実現されているものとする。以下、プログラミング言語として、Ｃ言語を例とする。このとき図８のように、モジュールＡからモジュールＢの関数ｆｕｎｃｔｉｏｎを呼び出す場合を考える。

【００１７】通常、ある関数を呼び出すときには引数が指定される。呼び出された関数では、この引数に応じて処理を行う。関数ｆｕｎｃｔｉｏｎの引数として、整数や浮動小数点などの定数、メモリ上の場所（アドレス）を指示すポインタ定数がある。通常はこれらの値を引数リストとして記述する。これを図９のように、一つのポインタ変数を引数とする、関数呼び出しを行うように書き換えることも可能である。

【００１８】具体的には関数の引数をすべて一つの構成要素にまとめる。整数や浮動小数点の値は、その値を構造体の構成要素とする。ポインタ変数の場合は、そのポインタ変数の指示す場所の値を、構造体の構成要素とする。関数を呼び出す側では、直接的・現実的な整数・浮動小数点・ポインタ変数の指示する場所の値を呼び出す。この実行行為が構造体の構成要件となっているため、汎用性が高い、類型的・形式的引数として迅速な処理行為をおこなうことが可能となる。

【００１９】呼び出された関数側では、この構造体のポインタが構造体の引数として渡されるので、構造体から引数を再構築し、本来の処理を行う。

【００２０】このように、もともとの引数を構造体にまとめ、構造体のポインタだけを引数として呼び出しすることで、本発明によってハードウェア化が可能となる。

【００２１】次に、モジュールの関数が複数ある場合を考える。図１０のようにモジュールＢにはｆｕｎｃｔｉｏｎ１とｆｕｎｃｔｉｏｎ２の２つがあるとする。ハードウェア化の観点から鑑みるとハードウェアインターフェースを１つに統一・共通化したほうが、回路規模を小さくできる場合がある。（このことは消費電力の低減やトランジスタの節約の観点からも重要であるといってよかろう。）この点、モジュールＡから呼び出される関数に対応するものを１つにすればよい。つまり関数の引数を一つに統一・共通化する実現方法として前述の構造体の構成要素として、本来呼び出したい関数の番号を内包する。（形式的な処理回路に実質的な処理を特別要件として処理するという点では刑法における団藤系保証人説的な意味合いの理解もしうる。）これにより、関数によって引数の構造体の定義が違っていたとしても、この番号を見ることによってどの構造体の定義が用いられてい

るかを判別することができる。逆にいえば、どのような種類の構造体が用いられており、その構造体を入力としてどの関数で処理を行えばよいかを関数番号があらわしていると考えることもできる。

【００２２】つまり図１１のように、モジュールＢでは、モジュールＡから呼び出される関数を、関数番号と構造体をポインタとする統一ｆｕｎｃｔｉｏｎで規定する。統一ｆｕｎｃｔｉｏｎでは、関数番号によってｆｕｎｃｔｉｏｎ１やｆｕｎｃｔｉｏｎ２を呼び出すことが可能である。

【００２３】これまでＣ言語を例にして説明してきたが、この点、Ｃ言語以外の言語においてもこれまでの本件特許の説明が適用できる。なぜならコンピュータの基礎理論である情報数学において、通常利用されているアセンブラ言語・ＢＡＳＩＣ言語・Ｃ言語・Ｃ＋＋言語・Ｊａｖａ言語等は各言語の記述能力が等価であることが証明されている。このことは、どの言語で書かれているソフトウェアでも他の言語に変換することが可能であることを述べており、本件における特許請求の説明が他の言語においても成り立つことを妨げない。

【００２４】次に本発明によるソフトウェアをハードウェア化する基本的手法を説明する。本発明におけるソフトウェアのモジュールは図１１に示す形をとっている必要がある。ここではモジュール間で引数構造が扱われているか説明する。図１２に例を示す。モジュールＢを見ると外部（この例ではモジュールＡ）からａｒｇｓに対応するメモリ領域が示され、この領域の情報を使って何らかの計算・処理を行う。モジュールＢも特別なモジュールＣを呼び出すことが一般的であるから、モジュールＣに対して、引数構造体のポインタを指定して関数を呼び出すことになる。つまり、モジュールの入力として引数構造体が渡され、モジュールはその構造体に書かれた情報から計算を行い、次のモジュールを呼び出す引数構造体を生成する。すべてのソフトウェアは、このような形式で実現することができる。

【００２５】本発明においてプログラミング言語で記述されたモジュールをハードウェア化する場合は、まず、ｆｕｎｃｔｉｏｎ１やｆｕｎｃｔｉｏｎ２の処理本体の関数部分をハードウェアで実現する。また引数構造体は、レジスタとしてハードウェアから参照することができる。つまり図１３のように、入力としての引数構造体に対応する入力レジスタと次のモジュール呼び出しのための引数構造体に対する出力レジスタがある。入力レジスタの値から、ハードウェアによって計算を行い、結果を出力レジスタに格納する。ハードウェアには、ヘッダの不完備化による情報処理装置を用いることも状態遷移機械を用いることも可能である。状態繊維機械の遷移や出力がプログラム可能なものとして、マイクロプロセッサがあるが、マイクロプロセッサによって入力レジスタから出力レジスタの値を計算することも可能である。

【００２６】次に、本発明によるハードウェアモジュール間の通信方法を説明する。本発明において重要なモジュール間通信システムの仕組みについて解説する。モジュール間の関数呼び出しにおける引数には、関数番号と引数構造体のポインタがあることを前節において述べた。計算ハードウェアから見ると、引数構造体の実態がレジスタとして参照できる必要があることから、実際はポインタではなく実体が必要である。つまり、モジュールをなすハードウェア間で、関数番号と引数構造体の実体をやり取りすればよい。

【００２７】図１４にモジュール間でやりとりするパケットの例を示す。関数番号と引数構造体からなるデータをパケットという塊にまとめ、このパケット単位で通信を行う。パケットの先頭１ワードには、関数番号が格納されており、この番号をもとにしてどの計算ハードウェアの入力レジスタに引数構造体を反映させるかを決定する。図１５には、パケットの入出力ハードウェアの例を示す。モジュールハードウェア間には、ユニバーサルプロトコルライン（Ｕ．Ｐ．Ｌ）とよばれるパケット通信があり、これによってモジュール間が接続される。Ｕ．Ｐ．Ｌは出力レジスタから入力レジスタへとその値を転送する仕組みである。Ｕ．Ｐ．Ｌの実装には、通常使われるシリアル転送の仕組みやＬＶＤＳ（Ｌｏｗ　Ｖｏｌｔａｇｅ　Ｄｉｆｆｅｒｅｎｔｉａｌ　Ｓｉｇｎａｌｉｎｇ）、３値論理方式など、転送速度、伝送距離などの諸条件にあったものを自由に選ぶことができる。

【００２８】出力レジスタは関数番号に相当するプロトコルコード部と引数構造体に相当するデータ部からなる。モジュールハードウェアによって、出力レジスタに値が設定されると、出力レジスタの値がＵＰＬの出力される。ＵＰＬによって受信されるパケットのうち、当該受信モジュールが受信すべきパケットだけを入力レジスタに設定する。受信すべきかどうかは、プロトコルコード部を参照することで判断できる。

【００２９】次に本発明応用例を示す。本発明によって，これまではソフトウェアで実現されていた様々なデータ処理装置を容易にハードウェア化することができる。次にその例を取り上げる。

【００３０】インターネットサーバ。インターネット上でサービスを提供するサーバといわれるコンピュータがある。インターネットでつながったクライアントコンピュータから様々な要求を受理し、その要求に応じてクライアントにデータを返すコンピュータである。このコンピュータには、オペレーティングシステムソフトウェアとサーバソフトウェアが動作している。インターネットが爆発的に拡大し、クライアントコンピュータ数が増加し、またアクセス回線がＡＤＳＬ・ＳＤＳＬ・光ファイバー等により、乗数倍的に高速化されているため、大量の要求がサーバコンピュータに集中的に負荷がかかっている。

【００３１】これまでは、サーバコンピュータのＣＰＵやメモリといったハードウェア的な処理能力を増強し、また、ソフトウェアの改良によって、処理可能な量を等差的に増大させてきた。しかしながら、もはや要求の増加のほうが処理能力を上回りつつあるし、現在すでに一時的に処理能力を上回ったため、サーバが機能できなくなり、サービスを停止してしまうということがしばしばみうけられる。

【００３２】そこで、本発明を情報処理装置に適用し、サーバ上で動作しているオペレーティングソフトウェアとサーバソフトウェアをハードウェアに置き換え、高速化を行う。

【００３３】サーバソフトウェアの例として、Ｗｅｂサーバのハードウェア化の例を図６に示す。（ネットワーク図、図１６参照）イーサネット（登録商標）受信モジュールによって、イーサネット物理層インタフェースが受信されたイーサパケットを処理し、Ｕ．Ｐ．Ｌに出力する。Ｕ．Ｐ．Ｌ上には、ＡＲＰ（アドレス解決プロトコル）を処理するＡＲＰモジュール、ＩＰ（インターネットプロトコル）を処理するＩＰモジュールがつながっている。イーサネットパケットのプロトコル識別子の値に応じて別々のプロトコルコードをつけられたパケットが生成される。これによって、イーサネットモジュールがＵＰＬに送信したパケットは、ＡＲＰモジュールが処理すべきかＩＰモジュールが処理すべきかが決定される。

【００３４】ＡＲＰモジュールは、ＡＲＰリクエストパケットを受信したときに、ＡＲＰリプライパケットを送信するという処理を担当するモジュールである。データ部に含まれるイーサネットパケットを参照しながら、リプライパケットを送信すべきかどうか、どういうないようのりプライパケットを送信すべきかをヘッダの不完備化による情報処理装置もしくはその他の回路によって決定し、出力レジスタに値を設定する。この出力レジスタは、Ｕ．Ｐ．Ｌによってイーサネット送信モジュールに接続されており、出力レジスタの値が送信できる状態になった時点で、イーサネット送信モジュールに向けて送信する。

【００３５】ＩＰモジュールでは、ＩＰパケットを受信したときに内容がＴＣＰモジュールへ、ＵＤＰパケットであればＵＤＰモジュールへといった分岐処理をおこなう。また、分断化されたＩＰパケットをもとに戻す処理、自分宛でないパケット処理なども適宜行う。ＴＣＰモジュールでは、ＴＣＰで定めたプロトコル処理を行う。ＨＴＴＰモジュールでは、ＨＴＴＰで定められたプロトコル処理を行う。

【００３６】コンテンツモジュールでは、Ｗｅｂサーバがクライアントコンピュータに送り返すべきＷｅｂデータを保持している。保持機構としては、フラッシュメモリ・スタティックメモリ等電気的記憶装置、ハードディスクといった固定記憶装置、このほかにも時期的記憶装置等、電気回路と接続可能な様々な記憶装置をもちいることができる。

【００３７】この他にもルータ、クライアントマシーンにおいても同様のことがあてはまる。

【図面の簡単な説明】

【図１】第Ｎ層における受信回路の説明図

【図２】ＯＳＩモデルと対応図

【図３】ＵＰＬの概念図

【図４】第Ｎ階層における送信回路図

【図５】入力ＵＰＬと出力ＵＰＬの説明図

【図６】本発明の全体図

【図７】ワードプロセッサの機能図

【図８】Ｃ言語を用いた関数呼び出し図

【図９】Ｃ言語をもちいたモジュール相関図

【図１０】Ｃ言語をもちいたモジュール相関図

【図１１】Ｃ言語をもちいたモジュール相関図

【図１２】計算（処理）によってパケットが生成される図

【図１３】パケットの生成の詳細な説明図

【図１４】関数番号と引数構造体の説明

【図１５】ＵＰＬの役割

【図１６】インターネットの概念図

【符号の説明】

１　ヘッダの完備化・不完備化による通信処理装置

２　Ｕ．Ｐ．Ｌ（Ｕｎｉｖｅｒｓａｌ　Ｐｒｏｔｏｃｏｌ　Ｌｉｎｅ）図

３　外部入力からの通信プロトコル情報を共通化・抽象化

４　内部処理における共通化・抽象化した不完備化した通信情報

５　ＯＳＩ７階層モデル

６　Ｕ．Ｐ．Ｌ（Ｕｎｉｖｅｒｓａｌ　Ｐｒｏｔｏｃｏｌ　Ｌｉｎｅ）図

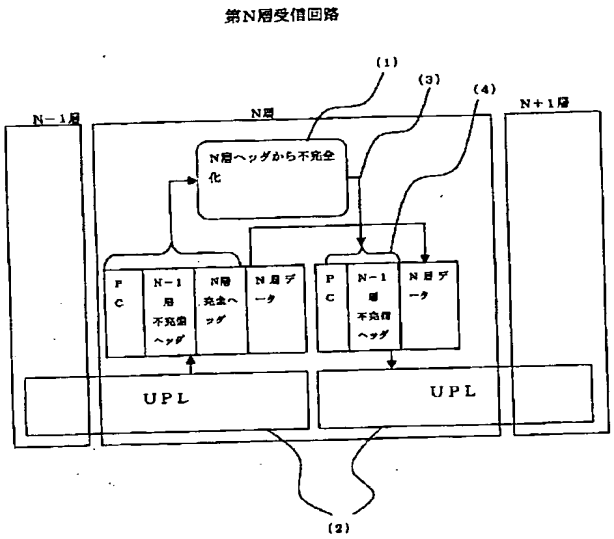
７　第Ｎ層送信回路

８　ＵＰＬ入力部

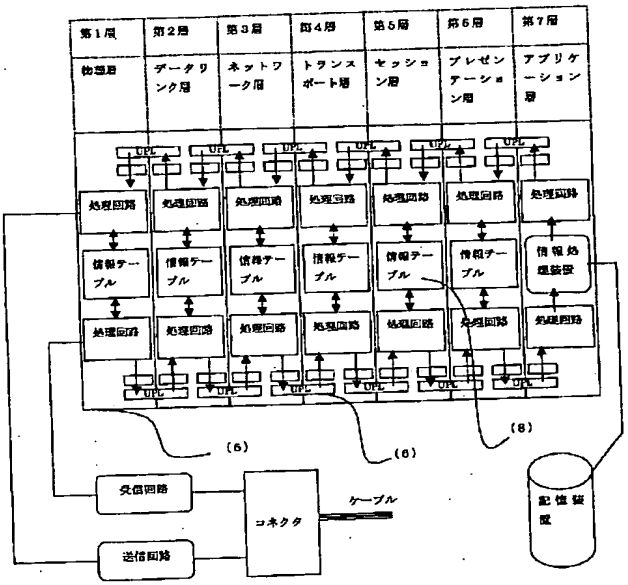
９　ＵＰＬ出力部

１０　記憶装置

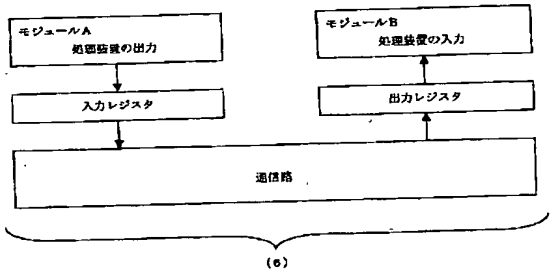
【図１】



【図２】

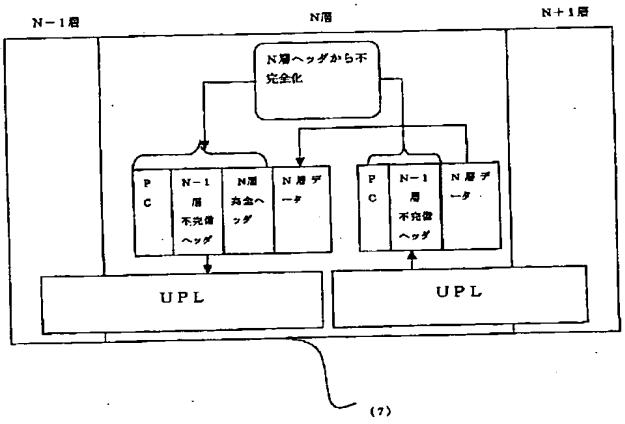


【図３】



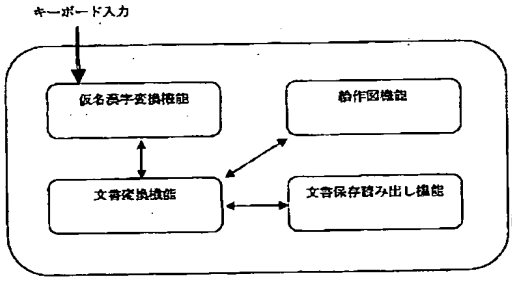
【図４】

第N層送信回路

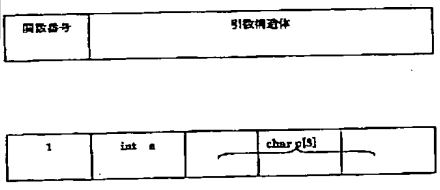


【図７】

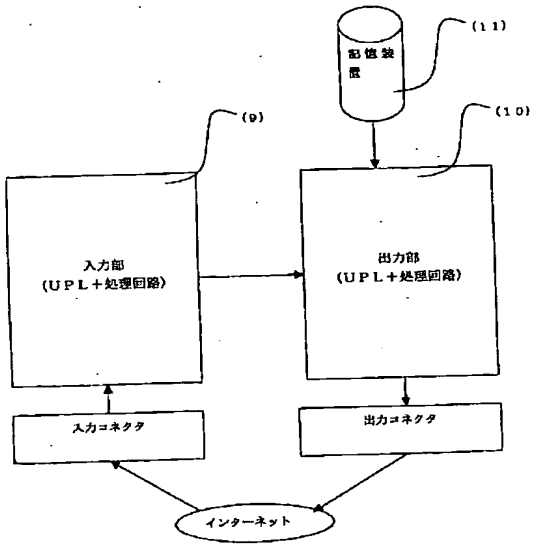
ワープロソフト



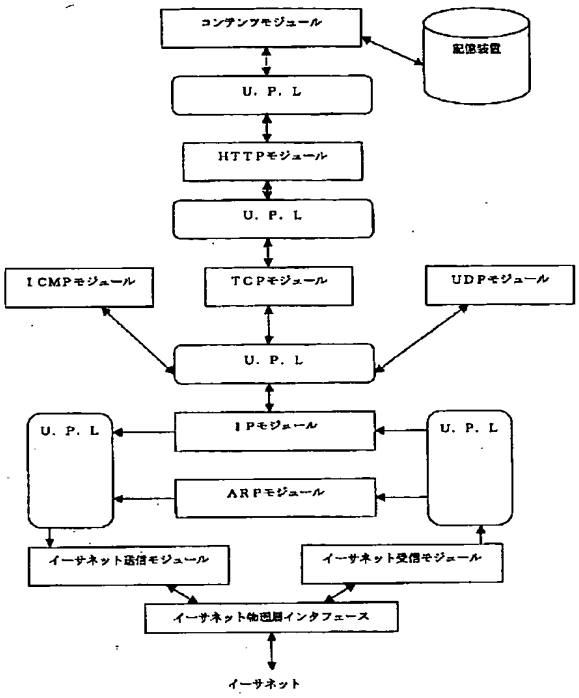
【図１４】



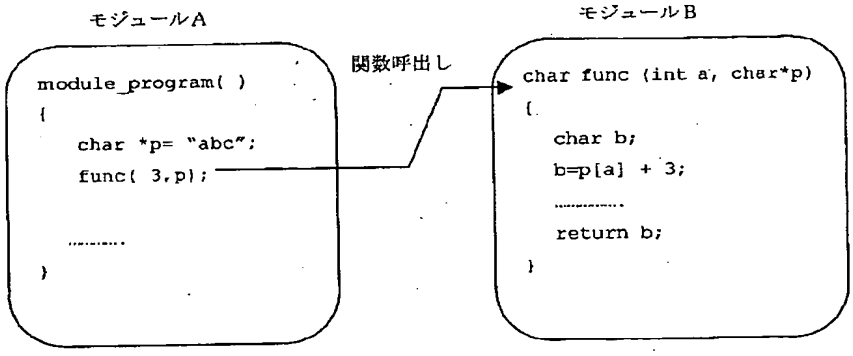
【図５】


記憶装置
(11)
(9)
(10)
入力部
(UPL＋処理回路)
出力部
(UPL＋処理回路)
入力コネクタ
出力コネクタ
インターネット

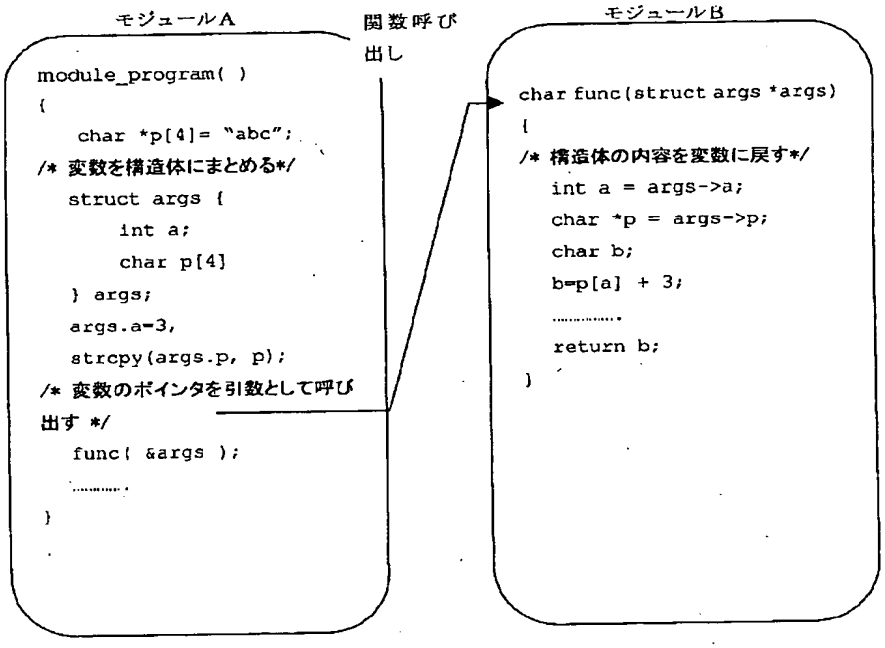
【図６】


コンテンツモジュール
記憶装置
U. P. L
HTTPモジュール
U. P. L
ICMPモジュール
TCPモジュール
UDPモジュール
U. P. L
U. P. L
IPモジュール
U. P. L
ARPモジュール
イーサネット送信モジュール
イーサネット受信モジュール
イーサネット物理層インタフェース
イーサネット

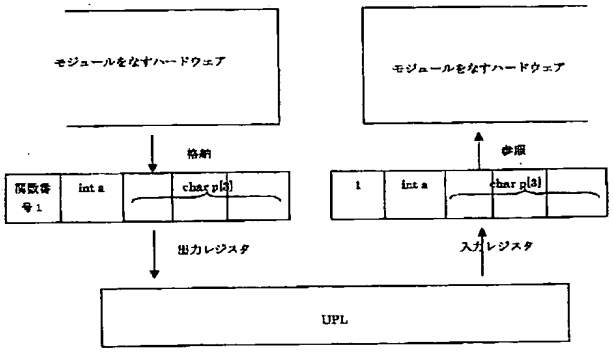
【図８】


モジュールA
モジュールB
関数呼出し
module_program( )
{
char *p= "abc";
func(3,p);
.....
}
char func (int a, char*p)
{
char b;
b=p[a] + 3;
.....
return b;
}

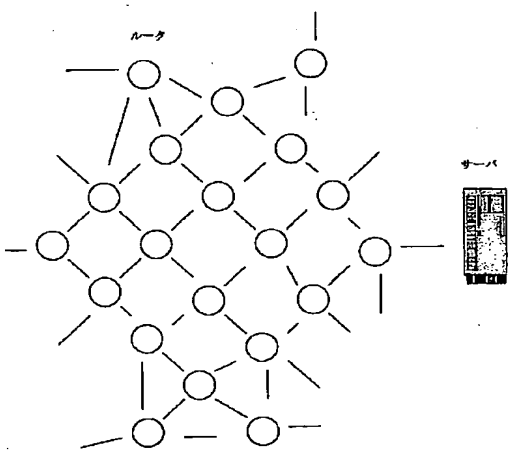
【図９】


モジュールA
関数呼び出し
モジュールB
module_program( )
{
char *p[4]= "abc";
/* 変数を構造体にまとめる*/
struct args {
int a;
char p[4]
} args;
args.a=3,
strcpy(args.p, p);
/* 変数のポインタを引数として呼び出す */
func(&args);
.....
}
char func(struct args *args)
{
/* 構造体の内容を変数に戻す*/
int a = args->a;
char *p = args->p;
char b;
b=p[a] + 3;
.....
return b;
}

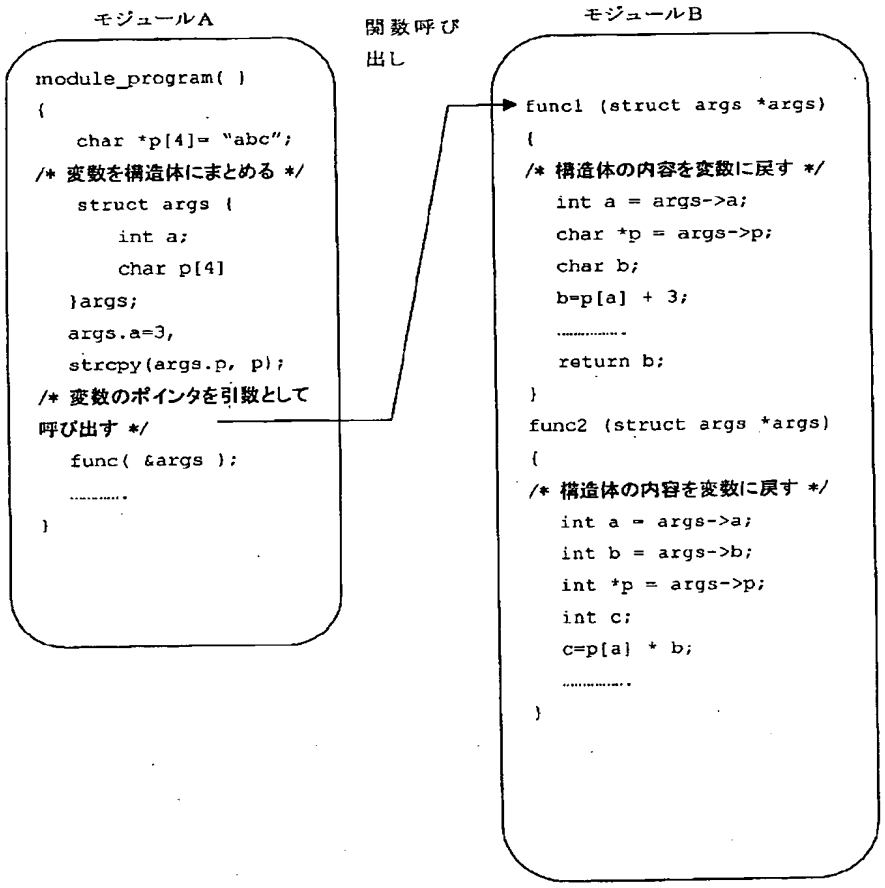
【図１５】


モジュールをなすハードウェア
モジュールをなすハードウェア
格納
参照
関数番号1
int a
char p[4]
1
int a
char p[4]
出力レジスタ
入力レジスタ
UPL

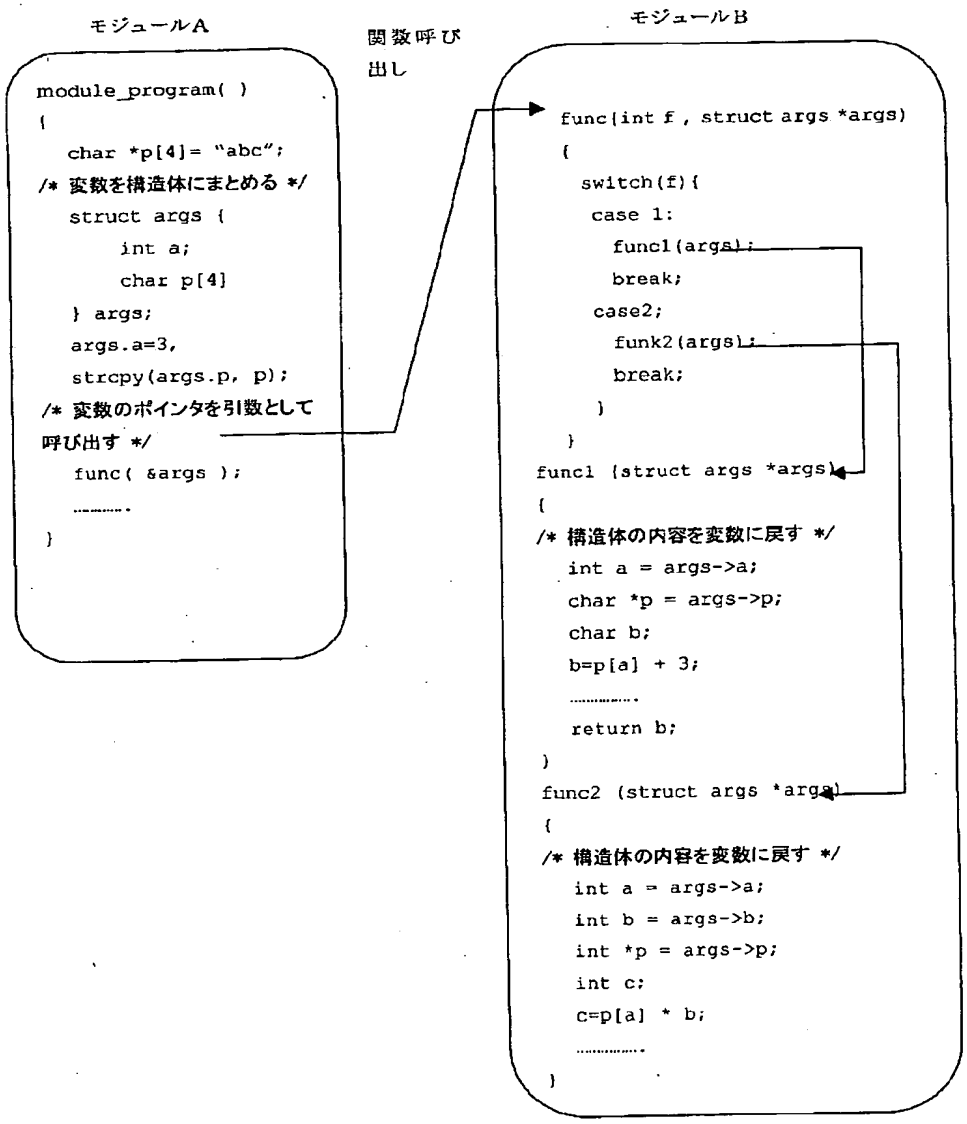
【図１６】


ルータ
サーバ

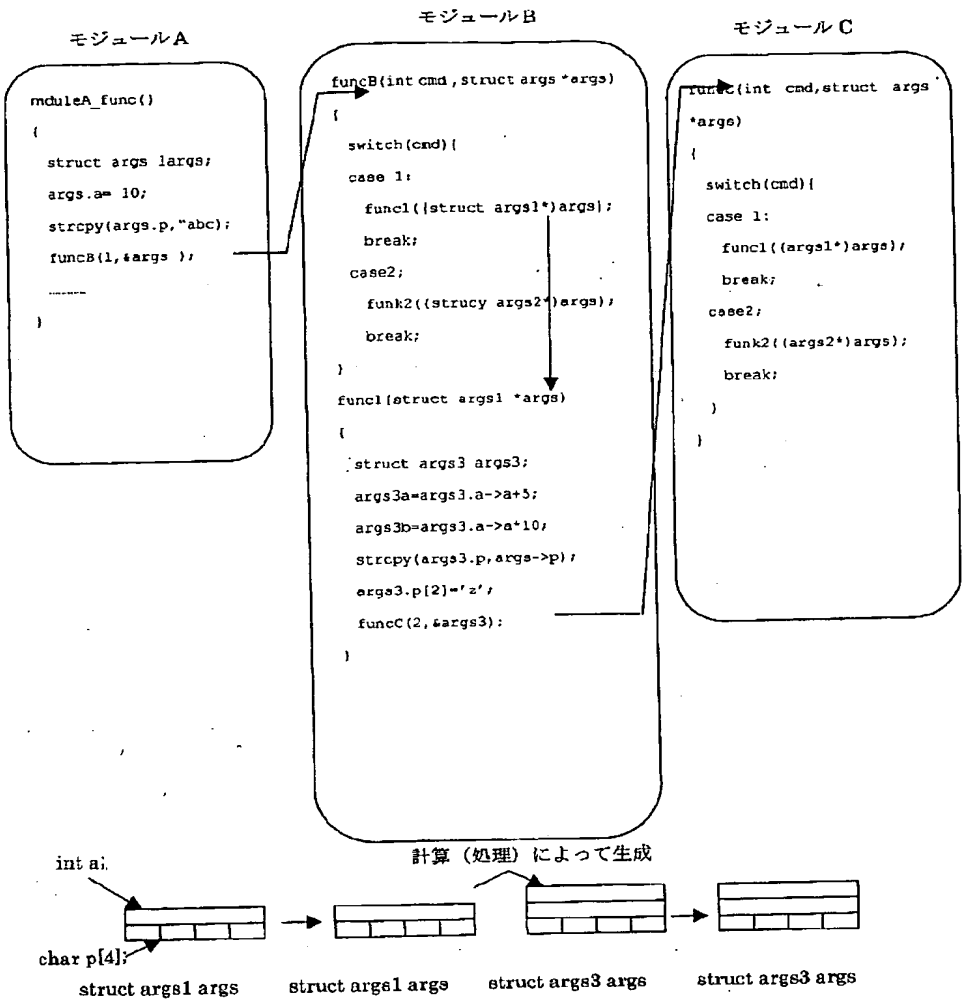
【図１０】


モジュールA
関数呼び出し
モジュールB
module_program( )
{
char *p[4]= "abc";
/* 変数を構造体にまとめる */
struct args {
int a;
char p[4]
}args;
args.a=3,
strcpy(args.p, p);
/* 変数のポインタを引数として呼び出す */
func(&args);
.....
}
func1 (struct args *args)
{
/* 構造体の内容を変数に戻す */
int a = args->a;
char *p = args->p;
char b;
b=p[a] + 3;
.....
return b;
}
func2 (struct args *args)
{
/* 構造体の内容を変数に戻す */
int a = args->a;
int b = args->b;
int *p = args->p;
int c;
c=p[a] * b;
.....
}

【図１１】


モジュールA
関数呼び出し
モジュールB
module_program( )
{
char *p[4]= "abc";
/* 変数を構造体にまとめる */
struct args {
int a;
char p[4]
} args;
args.a=3,
strcpy(args.p, p);
/* 変数のポインタを引数として呼び出す */
func(&args);
.....
}
func(int f , struct args *args)
{
switch(f){
case 1:
func1(args);
break;
case2;
funk2(args);
break;
}
}
func1 (struct args *args)
{
/* 構造体の内容を変数に戻す */
int a = args->a;
char *p = args->p;
char b;
b=p[a] + 3;
.....
return b;
}
func2 (struct args *args)
{
/* 構造体の内容を変数に戻す */
int a = args->a;
int b = args->b;
int *p = args->p;
int c;
c=p[a] * b;
.....
}

【図１２】


モジュールA
mduleA_func()
{
struct args largs;
args.a= 10;
strcpy(args.p,"abc);
funcB(1,&args);

}
モジュールB
funcB(int cmd,struct args *args)
{
switch(cmd){
case 1:
func1((struct args1*)args);
break;
case2;
funk2((strucy args2*)args);
break;
}
func1(struct args1 *args)
{
struct args3 args3;
args3a=args3.a->a+5;
args3b=args3.a->a*10;
strcpy(args3.p,args->p);
args3.p[2]='z';
funcC(2,&args3);
}
モジュールC
funcC(int cmd,struct args *args)
{
switch(cmd){
case 1:
func1((args1*)args);
break;
case2;
funk2((args2*)args);
break;
}
}
int a;
char p[4];
struct args1 args
struct args1 args
計算（処理）によって生成
struct args3 args
struct args3 args

【図１３】

```
func1(struct args1 *args)
{
    struct args3 args3;
    args3.a = args3.a->a+5;
    args3.b = args3.a->a*10;
    strcpy(args3.p,args->p);
    args3.p[2] = 'z';
    funcC( 2 , &args3 );
}
```